

# Inhaltsverzeichnis

Einleitung.....	3
Projektrandbedingungen.....	3
Teilnehmer.....	3
Zeitraum.....	4
Vorgehensmodell.....	4
Aufbau der Sprints.....	4
Internationaler Rahmen.....	5
Programmiersprachen.....	5
Anlagenbeschreibung.....	5
Beschreibung der aktuellen Anlage.....	5
Beschreibung der einzelnen Elemente.....	6
Stationen.....	6
Transportband.....	6
Schlitten.....	6
Paletten.....	6
Befehle.....	6
Netzwerkprotokoll.....	6
Anforderungen.....	7
Funktionale Anforderungen.....	7
Netzwerkverbindung.....	7
Interpretation der Nachrichten.....	7
Simulation der Nachrichten.....	9
Staubbehandlung.....	10
Fehlerbehandlung.....	10
Anzeige des Verbindungsstatus.....	11
Einstellen der Simulationsgeschwindigkeit.....	11
Anzeigen aller Stationen.....	12
Einfügen/ Löschen der Stationen.....	12
Konfiguration der Stationen.....	12
Erstellen von Kreuzungen.....	12
Log.....	12
Zustandsspeicherung.....	13
Status.....	13
Kontrolliertes Beenden.....	13
Wiederherstellung.....	13
Nicht-Funktionalen Anforderungen.....	14
Rechenzeit.....	14
Plattform.....	14
Nutzergruppe.....	14
Nutzerschnittstelle.....	14
Betriebszeit.....	14
Wartbarkeit.....	15
Maximale Anzahl an gleichzeitig vorhandenen Schlitten.....	15
Maximale Anzahl an gleichzeitigen Netzwerknachrichten.....	15
Anzahl Transportsysteme.....	15
Portierbarkeit.....	15
Architekturgrundlagen.....	16
Model-View-Controller.....	16
Klassendiagramm Model.....	16
Sequenzdiagramm Empfang einer Nachricht.....	17

Sequenzdiagramm Netzwerkschnittstelle.....	17
Layout.....	17
Gesamtüberblick.....	17
Optionen.....	19
Konfiguration der Stationen.....	19
Netzwerkkonfiguration.....	20
Station-Attribute.....	20

# **Einleitung**

Die Fabrik der Zukunft ist ein Teil des Forschungsprojektes Wirtschaft 4.0 im Mittelstand: Die digitale Transformation ("WiMiT")<sup>1</sup> und hat das Ziel, mittelständische Unternehmen in ihrer Arbeit effizienter zu gestalten und sie somit wettbewerbsfähiger zu machen. Mit dieser Motivation wird auch dieses Projekt geführt. Mit dem Ziel, die Überwachung der Abläufe des Transportes der FDZ<sup>2</sup> genauer und effizienter zu gestalten, wurde das Entwicklerteam beauftragt, eine Simulationssoftware für den Transport zu erschaffen.

In diesem Dokument wird das Vorgehen der Entwickler beschrieben, es wird auf die Rahmenbedingungen des Projektes eingegangen und klar definiert, welche Ziele und Anforderungen die Auftraggeber mit der Erschaffung dieses Projektes im Auge hatten. Zuerst wird das Entwicklerteam vorgestellt, anschließend die Rahmenbedingungen definiert, dann die Anforderungen technisch versiert festgehalten und zum Schluss erste Architektur- und Designentscheidungen dargestellt.

---

<sup>1</sup> Institut für Informationssysteme der Hochschule Hof

<sup>2</sup> FDZ, Fabrik der Zukunft

# Projektrahmbedingungen

Das Projekt „Simulation des Transports der Fabrik der Zukunft“ wird im Rahmen einer Studienarbeit im Fach Praktikum Softwareentwicklung in der Fakultät Informatik an der Hochschule Hof durchgeführt. Es steht unter der Leitung von Prof. Dr. René Peinl, der die Studienarbeit betreut, jedoch nicht das Projekt der FDZ<sup>3</sup>. Dazu mehr im nächsten Kapitel.

## Teilnehmer

Am Projekt sind im allgemeinen drei Gruppen tätig. Diese sind die Entwicklergruppe, bestehend aus den Studenten der Hochschule Hof, dem Studienarbeitsleiter Prof. Dr. René Peinl, und den Auftraggebern der FDZ. In der folgenden Tabelle werden alle Teilnehmer, sowie ihre Rollen im Projekt aufgelistet:

Gruppe	Name	Rolle
Studienarbeitsleiter	Prof. Dr. René Peinl	Beaufsichtigung und Beurteilung der Studienarbeitsleistungen der Studenten
Auftraggeber	Prof. Dr. Valentin Plenk	Beaufsichtigung der Leistungen im Rahmen der FDZ und Erstellen der Anforderungen
Auftraggeber	Alexander Schmid	Beaufsichtigung der Leistungen im Rahmen der FDZ, Rolle des Product-Owners im Vorgehensmodell Scrum, Priorisierung der Aufgaben
Entwickler	Andreas Glaser	Entwicklung und Dokumentation der Ergebnisse
Entwickler	Dzianis Brysiuk	Entwicklung und Dokumentation der Ergebnisse
Entwickler	Noah Lehmann	Entwicklung und Dokumentation der Ergebnisse

## Zeitraum

Das Projekt wird im Rahmen einer Studienarbeit im Fach Praktikum Softwareentwicklung durchgeführt. Dieses Fach wurde von den Entwicklern im 4. Fachsemester im Sommersemester 2018 belegt. Die Analyse der Anforderungen begann im März 2018, die Arbeiten an der Studienarbeit dauern bis Ende Juni. Eine gemeinsame Evaluation des Projektes findet am 05.07.2018 statt.

## Vorgehensmodell

Das Entwicklerteam hat sich zu Beginn des Projektes für das Vorgehensmodell Scrum entschieden. Dies hat sich aus den Gesprächen mit den Auftraggebern ergeben, welche von den Vorteilen einer engen Zusammenarbeit der Entwickler mit den Auftraggebern überzeugt waren.

---

<sup>3</sup> FDZ, Fabrik der Zukunft

## Aufbau der Sprints

Scrum ist eine agiles Vorgehensmodell, das zeitlich in verschiedene Sprints aufgeteilt wird. Da das Projekt „Fabrik der Zukunft“ selbst noch ein Prototyp ist, war es den Auftraggebern wichtig, Anforderungen immer neu einordnen zu können. Deshalb sind die einzelnen Sprints sehr kurz gehalten. Sprintreviews finden wöchentlich statt. In diesen werden die Aufgaben, die das Entwicklerteam im Laufe der Woche gefunden hat, neu priorisiert. Außerdem können so aktuelle Probleme und Inhalte, die im nächsten Sprint bearbeitet werden neu analysiert und besprochen werden, bevor sie umgesetzt werden.

Im folgenden wird der Ablauf eines Sprints kurz dargestellt:

### 1. Sammeln von Aufgaben

Vor jedem Sprint trifft sich das Entwicklerteam und sammelt Aufgaben, die erledigt werden müssen. Dabei spielt es keine Rolle, wann diese Aufgaben erledigt werden müssen. Alle Aufgaben werden in einem Sprint-Backlog<sup>4</sup> gesammelt.

### 2. Priorisierung der Aufgaben

Vor jedem Sprint trifft sich das Entwicklerteam mit dem Auftraggeber Alexander Schmid, ihm werden die gefundenen Aufgaben kurz erörtert und eventuelle Probleme dargestellt. Anschließend priorisiert er die Aufgaben.

### 3. Besprechung von Problemen

Das Team hat die Möglichkeit, unklare Anforderungen nochmals mit dem Auftraggeber zu besprechen und festzuhalten, bevor es die neuen Aufgaben bearbeitet.

### 4. Arbeitsphase

Das Team bearbeitet die Aufgaben möglichst nach der Reihenfolge, wie sie vom Auftraggeber priorisiert wurden. Abweichungen sollten vorher beim Meeting angemerkt und dem Auftraggeber erläutert werden.

### 5. Evaluation

Die letzte Phase des Sprints ist das Sprint Review, hier werden dem Auftraggeber die Ergebnisse des Sprints gezeigt. Er hat dann im nächsten Sprint die Möglichkeit, seine Prioritäten neu zu vergeben. Damit endet ein Sprint.

## Internationaler Rahmen

Auf Wunsch der Auftraggeber wird das Programm, also der Quellcode und die Oberfläche, in Englischer Sprache verfasst. Dies hat den Zweck, dass sowohl internationale Firmen sich von der FDZ inspirieren lassen können, aber auch internationale Studenten im späteren Verlauf der FDZ an dem Programm mitwirken, beziehungsweise es pflegen und warten können.

---

<sup>4</sup> Ein Sprint-Backlog wird in einem separaten Dokument geführt und am Ende des Projektes mit abgegeben.

# **Programmiersprachen**

Nach Vorgabe der Auftraggeber wird das Programm in der Programmiersprache Java implementiert. Die genaue Version wurde festgelegt auf Java 1.8, da dies die Version ist, in der alle anderen Komponenten der FDZ programmiert wurden.

Die grafische Benutzeroberfläche wird in JavaFX implementiert. Dies ist aus dem selben Grund festgelegt worden, wie die eigentlich Programmiersprache Java.

Insofern in diesem Projekt weitere Programmiersprachen oder Frameworks und Bibliotheken genutzt werden, so wird dies jeweils in eigenen Analysen und Dokumentationsabschnitten festgehalten.

# **Anlagenbeschreibung**

In diesem Kapitel wird der Aufbau der Fabrik der Zukunft kurz erläutert. Dies ist für den weiteren Verlauf der Analyse und des Pflichtenheftes insofern wichtig, da die folgenden Kapitel auf dem Verständnis über das gegebene Projekt aufbauen.

## **Beschreibung der aktuellen Anlage**

Die Fabrik der Zukunft, im Folgenden nur noch FDZ<sup>5</sup> genannt, ist ein Forschungsprojekt des IISYS<sup>6</sup>, ein Forschungsinstitut der Hochschule für Angewandte Wissenschaften in Hof. Sie fällt in die Kategorie der Industrie 4.0, welche für die Digitalisierung und Automatisierung im Bereich der Produktion steht.

Die FDZ ist eine Fabriksimulation, welche anhand von echten Elementen, wie z.B. einem Roboter, einer Ein-/Ausgabe und einem Lager, auch genannt Stationen, den echten Ablauf einer Produktion nachstellen soll. Das Ziel ist es, automatisiert auf verschiedene Anfragen der Kunden zu reagieren, denn die Produktion funktioniert ohne Eingreifen einer natürlichen Person. Nach Aufgabe einer Bestellung kann ein Transportband Ware über Paletten und Schlitten zwischen den Stationen verschieben, bei jedem Halt wird die Ware dann manipuliert, gelagert oder ausgegeben.

## **Beschreibung der einzelnen Elemente**

Die FDZ besteht, wie oben beschrieben, aus verschiedenen Elementen. Die für dieses Projekt relevanten Elemente werden im Folgenden genauer beschrieben.

### **Stationen**

Eine Station ist ein Teil der Produktion. Sie führt eine Manipulation an der Ware durch, lagert diese oder kann neue Teile der Ware entgegennehmen oder ausgeben. In der aktuellen Zusammensetzung der FDZ gibt es 3 Stationen, eine Ein-/Ausgabe, die neue Schlitten und Paletten entgegen nimmt, ein Lager, welches die Paletten mit Inhalt lagern kann, und einen Roboter, welcher die Zusammensetzung der Ware auf den Paletten manipulieren kann.

---

5 FDZ, Fabrik der Zukunft

6 Institut für Informationssysteme der Hochschule Hof

## **Transportband**

Das Transportband ist das zentrale Element in der FDZ. Es verbindet die einzelnen Stationen miteinander und befördert Schlitten zu den benötigten Positionen.

## **Schlitten**

Ein Schlitten ist eine Konstruktion, welche die Paletten mit ihrem Inhalt auf dem Transportband befördert.

## **Paletten**

Die Paletten sind die Elemente, welche die eigentliche Ware beinhalten. Sie können z.B. vom Lager aufgenommen oder vom Roboter vom Transportband genommen werden. Paletten benötigen einen Schlitten unter sich, um auf dem Transportband bewegt werden zu können. Sie werden zum Ansprechen von der Software immer mit einer ID<sup>7</sup> gekennzeichnet

## **Befehle**

In der FDZ gibt es viele verschiedene Software Komponenten, die miteinander kommunizieren müssen. Dies können sie über das Netzwerk, an dem die Computer mit der Software angeschlossen sind. Die einzelnen Komponenten tauschen Befehle aus, um Reaktionen der Fabrik auszulösen.

## **Netzwerkprotokoll**

Bei dem Entwurf der FDZ wurde ein Netzwerkprotokoll eingeführt, das eigens für diesen Zweck entworfen wurde. Es beinhaltet die Befehle, welche über das TCP/IP Protokoll zu den einzelnen Softwarekomponenten gesendet werden.<sup>8</sup>

# **Anforderungen**

Dieses Projekt befasst sich mit der Simulation des Transports der FDZ. Der Transport ist der Teil der Fabrik, welcher sich mit der Verschiebung und Positionierung der Paletten befasst. Die fertige Software soll im Allgemeinen einen Überblick über die aktuell Konfiguration der Fabrik, die Positionen der einzelnen Paletten und Probleme in der Ausführung der empfangenen Transport-Befehle anzeigen. Im Folgenden werden die Anforderungen, getrennt in funktionale und nicht-funktionale Anforderungen, die an das System gestellt wurden, beschrieben.

## **Funktionale Anforderungen**

In diesem Teil des Kapitels geht es um die Anforderungen, die eine konkrete Funktion der Software beschreiben. Qualitative Anforderungen werden im zweiten Teil dargestellt.

---

<sup>7</sup> ID – Ganzzahliger Kennzeichner

<sup>8</sup> Siehe DocumentationNetworkCommands.pdf

## Netzwerkverbindung

Um die Befehle an den Transport der Fabrik empfangen zu können, muss das Programm eine Netzwerkverbindung zu einer übergeordneten Komponente der FDZ aufbauen können. Das Netzwerkprotokoll, welches zur Versendung der Befehle genutzt wird, ist das TCP/IP Protokoll.

Die weitere Analyse wird zum Zeitpunkt der Bearbeitung in einem separaten Dokument aufgeführt.

## Interpretation der Nachrichten

Sobald der Thread, der auf Nachrichten aus der Netzwerkverbindung hört, eine solche empfangen hat, startet er einen neuen Thread, der als Kommando-Interpreter fungiert, aus. Diesem wird als Konstruktor-Parameter die komplette Netzwerknachricht als String übergeben. Danach wird die Nachricht in den folgenden Schritten geparsed:

### 1. Parsen der Kommando-Nummer:

Jedes Kommando startet mit der Zeichenfolge STStK, gefolgt von einer dreistelligen Zahlenfolge, welche die Art des Kommandos beschreibt. Die Zahlenfolge kann wie folgt aussehen:

Kommando	Beschreibung	Anzahl Parameter
001	Anfordern eines leeren Schlittens zu angegebenen Position	1
002	Freigeben eines Schlitten, der die Palette mit einer angegebenen ID trägt	1
003	Verschieben eines Schlittens mit einer Palette mit der angegebenen ID zur angegebenen Position	2
004	Transport herunterfahren	0

Ist die Zahlenfolge nicht bekannt, so wird der übergeordneten Komponente der FDZ die Nachricht „Command not understood“<sup>9</sup> geschickt.

### 2. Validieren der Anzahl an mitgegebenen Parametern:

Wie in der obigen Tabelle zu sehen ist, hat jedes Kommando eine definierte Anzahl an Parametern. Ein Teil der Netzwerknachricht beschreibt die folgende Anzahl an Parametern. Dies sieht in der Nachricht wie folgt aus: StSTK002<Message-ID>**0002**xx.

Die fett markierte Zahlenfolge beschreibt die Anzahl der folgenden Parameter. In diesem Schritt des Parsens wird die Anzahl der Parameter mit der erwarteten Anzahl überprüft. Sollte diese nicht übereinstimmen, so wird der übergeordneten Komponente der FDZ die Nachricht „Command not understood“<sup>10</sup> geschickt.

<sup>9</sup> Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

<sup>10</sup> Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

### **3. Validieren des Aufbaus der Nachricht**

Bevor das Kommando weiter geparsed wird, wird überprüft, ob die Nachricht dem FDZ-Netzwerkprotokoll<sup>11</sup> entsprechend aufgebaut ist. Bei Fehlern im Aufbau der Nachricht wird angenommen, dass diese nicht fehlerfrei interpretiert werden kann. Deshalb wird der übergeordneten Komponente der FDZ die Nachricht „Command not understood“<sup>12</sup> geschickt.

### **4. Parsen der übergebenen Positionen**

Der erste Schritt beim finden der Parameter ist das Suchen der übergebenen Position. In der passenden Methode wird erst abgeglichen, ob das erwartete Kommando eine Position erwartet. Sollte das nicht der Fall sein, wird dieser Schritt übersprungen. Ansonsten wird die Position, die immer an letzter Stelle in der Nachricht durch eine zweistellige Abkürzung des Stationsnamens übergeben wird, ausgelesen und abgespeichert. Ist keine Position auffindbar, so wird der übergeordneten Komponente der FDZ die Nachricht „Command not understood“<sup>13</sup> geschickt.

### **5. Parsen der übergebenen Paletten-ID**

Der nächste Schritt ist ähnlich aufgebaut, wie das Parsen der Position. In der passenden Methode wird zuerst überprüft, ob eine Paletten-ID in dem erwarteten Kommando benötigt ist. Danach wird die ID ausgelesen und abgespeichert. Sollte keine ID auffindbar sein, so wird der übergeordneten Komponente der FDZ die Nachricht „Command not understood“<sup>14</sup> geschickt.

### **6. Parsen der Message-ID**

Der letzte Schritt ist das auslesen der Message-ID<sup>15</sup>. Diese ist immer eindeutig aus dem Zeitpunkt des Abschickens zusammengesetzt und ist in der Nachricht nach der Kommandonummer eingefügt. Diese wird nicht auf ihre Korrektheit überprüft, da sie auf die Interpretation der Kommandos keinen Einfluss hat, solange alle anderen Parameter vollständig sind.

### **7. Validieren der Parameter**

Der vorletzte Schritt überprüft die gespeicherten Parameter auf ihre Korrektheit. Damit ist gemeint, ob sie in den zulässigen Wertebereichen liegen.

### **8. Erzeugen eines Kommando-Ausführers**

Anhand der Kommandonummer wird nun der passende Kommando-Ausführer erzeugt. Es gibt für jedes Kommando einen eigenen Konstruktor mit den passenden Übergabeparametern.

---

11 Siehe DocumentationNetworkCommands.pdf

12 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

13 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

14 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

15 Siehe FDZ Dokumentation, Spec.pdf

## Simulation der Nachrichten

Wie oben beschrieben gibt es für die Simulation der Nachrichten eine eigene Klasse. Diese wird `CommandExecutor` heißen. Sie wird vier verschiedene Konstruktoren haben, die jeweils die Übergabeparameter für ein bestimmtes Kommando benötigen. Die Übergabeparameter werden nicht auf ihre Korrektheit überprüft, da dies im Kommando-Interpreter schon geschehen ist. Jeder Konstruktor ruft lediglich die zum Kommando passende Methode auf. Diese Methoden sind wie folgt aufgebaut:

- **Anfordern einer neuen Palette zu Station xx**

Übergebene Parameter	Schritte
Position	<ol style="list-style-type: none"><li>1. Suchen der Position in bekannten Stationen. Wenn Station nicht bekannt ist, wird „Command not Executed“ an übergeordnete Komponente der FDZ geschickt<sup>16</sup>.</li><li>2. Überprüfen, ob Station bereits belegt ist oder ein Stau auf dem Weg zu ihr bekannt ist. Wenn nicht, dann Schlitten in Station vermerken. Falls belegt ist, Staubehandlung<sup>17</sup>.</li></ol>

- **Freigeben einer definierten Palette**

Übergebene Parameter	Schritte
Paletten-ID	<ol style="list-style-type: none"><li>1. Suche der passenden Palette in Listen der Stationen.</li><li>2. Falls gefunden, Entfernen der Palette aus Station, falls nicht gefunden, ist die Palette entweder im Stau oder nicht vorhanden.</li><li>3. Ist ein Schlitten im System, der leer ist, so wird ihm die unbekannte ID zugeteilt und er wird freigegeben.</li></ol>

- **Neupositionierung einer definierten Palette zu gegebener Station**

Übergebene Parameter	Schritte
Paletten-ID, Position	<ol style="list-style-type: none"><li>1. Suchen der Position in Liste der Stationen.</li><li>2. Überprüfen, ob ein Stau an der Station oder auf dem Weg zur ihr besteht.</li><li>3. Suchen der ID in Listen der Stationen. Ist sie bekannt wird sie zu neuer Station zugeordnet.</li><li>4. Ist die ID unbekannt, so wird überprüft, ob ein leerer Schlitten im System ist. Ihm wird die ID zugeteilt, danach wird er zur Position verschoben.</li></ol>

16 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

17 Wird später beschrieben

- **Herunterfahren des Transportes**

Übergebene Parameter	Schritte
keine	1. Abfrage, ob Programm geschlossen werden soll.

## Staubbehandlung

Wie in dem Kapitel „Simulation der Nachrichten“ beschrieben, kann es an Stationen zu einer Staubbildung kommen. Wenn eine Station mit einem Schlitten belegt ist, kann es dazu kommen, dass diese Station ein Kommando stört. Sie könnte entweder auf dem Weg eines Schlittens zu einer anderen Station liegen oder selbst das Ziel eines Schlittens sein. Die Bildung eines Staus kann wie im obigen Kapitel beschrieben erkannt werden. Die Behandlung eines Staus bedarf jedoch weiterer Analyse, die zum Zeitpunkt des Bearbeitens im Team erörtert werden muss<sup>18</sup>. Bis jetzt steht nur fest, dass eine optische Rückmeldung für den Nutzer ausgegeben wird, die symbolisiert wurde, wo ein Stau erkannt wurde. Außerdem

## Fehlerbehandlung

Wie aus den vorherigen Kapiteln klar hervorgeht, kann es an vielen Stellen vorkommen, dass Fehler auftreten, die ohne passende Behandlung zum Systemabsturz führen. Folgende Punkte müssen für dieses Projekt gesondert betrachtet werden:

- **Abbruch der Netzwerkverbindung**

Die Netzwerkverbindung kann an drei verschiedenen Stellen abbrechen. Der einfachste Fall ist der, wenn momentan keine Nachricht bearbeitet wird. An dieser Stelle muss nur versucht werden, die Verbindung wieder aufzubauen. Der zweite Fall tritt ein, wenn die Verbindung nach Eingang einer Nachricht abbricht. Jetzt darf zwar die Nachricht interpretiert werden, aber nicht simuliert werden, solange nicht ACK1<sup>19</sup> an die übergeordnete Komponente geschickt wurde. Der letzte Fall beschreibt die Situation, wenn die Verbindung abbricht, bevor ACK2<sup>20</sup> versendet wurde. Dies betrifft den Ablauf des Systems nicht, es muss nur beachtet werden, dass die Verbindung wieder aufgebaut werden muss, um das Acknowledgement noch zu versenden.

- **Empfangen einer Fehlerhaften Nachricht**

Wie oben beschrieben werden die Nachrichten vom Interpreter immer auf ihre korrekte Zusammensetzung, orientiert am Netzwerkprotokoll der FDZ, überprüft. Wird dort ein Fehler gefunden, so wird eine `IllegalCommandException` geworfen. Wird diese gefangen, muss eine Payload erstellt werden, die das Problem kurz beschreibt. Diese Payload wird dann mit einem „Command not Understood<sup>21</sup>“ Fehler an die übergeordnete Instanz der FDZ gesendet.

---

18 Nicht analysierte Punkte werden in gesonderten Dokumenten behandelt

19 Siehe DocumentationNetworkCommands.pdf

20 Siehe DocumentationNetworkCommands.pdf

21 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

- **Fehler in Zustand der Fabriksimulation**

Auch wenn die empfangene Nachricht korrekt verfasst wurde, kann es dazu kommen, dass ein Fehler ausgelöst wurde, der von Interpreter nicht abgefangen werden konnte. Das kann entweder eine nicht vorhandene Paletten-ID oder Station, oder eine Staubildung sein.

Sollte die Paletten-ID nicht bekannt sein, so wird überprüft, ob ein leerer Schlitten vorhanden ist. Ist dies der Fall, so bekommt er die unbekannte ID. Trifft dies nicht zu, so wird ein „Command not Executed<sup>22</sup>“ Fehler an die übergeordnete Instanz der Fabrik gesendet.

Sollte die Station nicht bekannt sein, so wird ein „Command not Executed<sup>23</sup>“ Fehler an die übergeordnete Instanz der Fabrik gesendet.

Sollte ein Stau gefunden werden, so wird nach dem Kapitel Staubbehandlung gehandelt.

Bei jeder Art von Fehler wird der Fehler an sich auch, wie im Kapitel Log beschrieben, festgehalten.

## Anzeige des Verbindungsstatus

Am linken Rand des Simulationsbildschirms soll eine Darstellung<sup>24</sup> anzeigen, ob eine Verbindung zur übergeordneten Komponente der Fabrik aufgebaut ist. Diese Darstellung soll die zwei Elemente der Fabrik darstellen, die passenden IP und Port Konfigurationen zu den Komponenten anzeigen und symbolisieren, ob eine Verbindung besteht oder nicht.

## Einstellen der Simulationsgeschwindigkeit

Der Nutzer soll die Möglichkeit haben, die Geschwindigkeit der Simulation einzustellen. Hierzu hat er zwei Optionen:

1. Betätigen eines Toggle-Buttons der die Simulationsgeschwindigkeit auf die Geschwindigkeit erhöht, welche der Prozessor für die Berechnungen des Programms benötigt.
2. Der Benutzer hat die Möglichkeit, für jede einzelne Verbindung die Zeit in Sekunden einzutragen, die er für passend hält. Hier bietet sich die Zeit an, die ein Schlitten in der realen FDZ-Umgebung für eine Verbindung benötigt.

---

22 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

23 Siehe DocumentationNetworkCommands.pdf, Kapitel 4.1.1 Common Errors

24 Siehe Kapitel Screendesign im Anhang

## Anzeigen aller Stationen

Das Simulationsprogramm soll die Stationen alle auf einem Blick anzeigen<sup>25</sup>. Diese werden als Quadrat auf dem Bildschirm ausgegeben. Es werden folgende Attribute der Stationen angezeigt:

- **Name:** Name der Stationen.
- **Stau:** Entweder Stau angezeigt, dazu eine Liste der Paletten-IDs im Stau, oder Station ist frei.
- **Status:** Entweder Station ist leer oder mit einer Paletten-ID besetzt.

## Einfügen/ Löschen der Stationen

Der Nutzer muss die Möglichkeit haben, Stationen einzufügen und zu löschen. Beim einfügen muss er alle Attribute der Station setzen können. Durch Löschen einer Station müssen alle Verbindungen zu ihr automatisch mit entfernt werden.

## Konfiguration der Stationen

Die Attribute der Stationen müssen jederzeit veränderbar sein. Die wichtigsten Attribute sind die Verbindungen zwischen den Stationen. Der Anwender muss bei jeder Station angeben, von welcher anderen Station sie erreichbar ist. Er tut dies durch bestätigen einer Check-Box. Wird die Check-Box nochmal betätigt, so verschwindet die zugehörige Verbindung und das Textfeld wird wieder funktionsunfähig gemacht.

Anzumerken ist, das jede Station nur einen Eingang und einen Ausgang hat.

## Erstellen von Kreuzungen

Zur besseren optischen Darstellung wurden Kreuzungen eingefügt. Der Auftraggeber schätzt es Realitätsnäher ein, dass eine Station nur einen Eingang und einen Ausgang hat. Sollte eine Station von mehreren anderen Stationen erreichbar sein, so muss dies durch Kreuzungen realisiert werden. Diese haben **n** Eingänge und **m** Ausgänge, mit  $n,m > 0$ . Verbindungen von und zu Kreuzungen werden genauso wie in Stationen erstellt.

---

25 Siehe Kapitel Screendesign im Anhang

## Log

Das Vorgehen des Systems muss zu jeder Zeit dokumentiert werden, um bei einem unerwarteten Fehler auch nachvollzogen werden zu können. Das genaue Vorgehen des Loggens kann zu diesem Zeitpunkt noch nicht beschrieben werden<sup>26</sup>. Es steht aber bereits fest, dass folgende Elemente sicher geloggt werden müssen:

- Einkommende Befehle anzeigen/speichern
- Interpretation anzeigen/speichern
- Reaktion anzeigen/speichern
- Abarbeitung anzeigen/speichern
- Staumeldung
- Fehler/ Exceptions (schwerwiegend/ fatal)

Die Logs werden sowohl in einer externen Log-Datei gespeichert, als auch auf dem Bildschirm zur Laufzeit ausgegeben.

## Zustandsspeicherung

Bei einem Systemabsturz muss das Programm in der Lage sein, seinen Zustand nach dem Neustart wiederherzustellen. Beim Neustart nach einem Absturz wird dem Nutzer die Frage gestellt, ob er den Zustand vor dem Absturz rekonstruieren möchte.

Änderungen im Zustand werden sofort nachdem sie eingetroffen sind in eine JSON<sup>27</sup> Datei abgespeichert<sup>28</sup>. Wird das System kontrolliert beendet, so wird die Datei gelöscht. Bei jedem Systemstart wird dann überprüft, ob die Datei vorhanden ist oder nicht. Ist sie vorhanden, so kann man daraus schließen, dass ein Absturz erfolgt ist und der Anwender bekommt die Möglichkeit, den Zustand der Datei zu rekonstruieren.

## Status

Neben dem Log soll auf dem Bildschirm zur Laufzeit auch der aktuelle Status des Programms ausgegeben werden. Wichtig ist, dass der Status des Programms ausgegeben wird, nicht der der Fabrik.

Die Statusmeldungen beinhalten, was das Programm im Moment der Anzeige bearbeitet. Diese Aktion wird in menschenlesbarem Text ausgegeben. Beispielhafte Nachrichten können z.B. sein:

- „*Interpretieren der Nachricht STstK00102329120002ro*“
- „*Ausführen der Nachricht ,Reposition Carriage 03 to Stock*“

---

26 Wird in einem gesonderten Dokument analysiert

27 Javascript-Object-Notation

28 Zu dem Speichervorgang an sich wird es ein gesondertes Dokument geben

## Kontrolliertes Beenden

Wie bereits beschrieben wird beim kontrollierten Beenden des Programms die Zustands-Speicherdatei gelöscht. So kann bei einem Neustart analysiert werden, ob beim letzten Programmablauf ein Absturz erfolgt ist.

## Wiederherstellung

Bei einem Neustart wird nach der Zustands-Speicherdatei gesucht. Wird diese nicht gefunden, so wird davon ausgegangen, dass das Programm kontrolliert beendet wurde. Wird jedoch eine gefunden, so bekommt der Nutzer in einem Abfragedialog die Möglichkeit, die Zustands-Speicherdatei einzulesen oder sie zu verwerfen.

## Nicht-Funktionalen Anforderungen

In diesem Kapitel werden die Anforderungen beschrieben, die keine echte Funktion beschreiben, sondern eher durch eine qualitative Eigenschaft des Programms definiert werden.

## Rechenzeit

Im folgenden Kapitel werden die wichtigsten Aktionen im System beschrieben. Es wird festgelegt, wie lange das System rechnen darf, bevor es die Aktion ausgeführt haben muss. Folgende Aktionen werden beschränkt:

Aktion	Zeit (in Sekunden)
Systemstart	<10
Konfiguration laden	<5
Konfiguration speichern	<5
Wiederherstellung	<7
Interpretation eines Befehls	<0,5
Ausführung eines Befehls	<0,5
Änderung in der Darstellung (z.B. neue Station einfügen, neue Verbindung)	<1
Beenden des Systems	<2

## **Plattform**

Nach Vorgabe des Kunden soll das fertige Simulationsprogramm auf einem Standard-PC laufen, auf dem das Betriebssystem Windows 7 installiert ist. Die Mindestanforderung an diesen PC sind folgende:

- Mindestens ein 2-Kern Prozessor
- Mindestens 1GB Arbeitsspeicher
- Bildschirmauflösung 1280x1024 oder besser
- Java Version 1.8 installiert
- Netzwerkschnittstelle

## **Nutzergruppe**

Die Nutzergruppe kann in zwei weitere Gruppen unterteilt werden. Die erste Gruppe besteht aus den Testern und Entwicklern, sowie den Zuständigen Verwaltern der Fabrik der Zukunft. Diese nutzen das Programm, um andere Elemente der Fabrik zu testen.

Die zweite Nutzergruppe besteht aus möglichen Kunden des IISYS<sup>29</sup>, diese haben die Möglichkeit, sich von den leitenden Informatikern beraten zu lassen, die FDZ zu testen und die Grundideen so auf ihre eigenen Betriebe zu übernehmen.

## **Nutzerschnittstelle**

Die Schnittstelle, die den Nutzern zum Bedienen der Simulation bereit gestellt werden muss, ist das Standard-Zubehör eines PCs. Dazu gehört eine Maus, um die Elemente der grafischen Oberfläche zu verschieben und zu betätigen, eine Tastatur, um gewünschte Eingaben zu tätigen und allen voran, wie im Kapitel Plattform beschrieben, ein Bildschirm, um die Simulation auch optisch erfassen zu können.

## **Betriebszeit**

Die Anforderungen an die Betriebszeit sind klar definiert. Obwohl dies im laufenden Betrieb vorerst nicht geplant ist, muss das Simulationsprogramm jederzeit abrufbar sein. Es muss auch in der Lage sein, über längere Zeitraum, theoretisch unbegrenzt lang, in Betrieb zu sein.

## **Wartbarkeit**

Ein wichtiger Punkt in der Wartbarkeit ist das modulare Aufbauen der Einzelteile der Simulation. Das Programm wird in drei Teile zerlegt, einmal die Logik, dazu die grafische Oberfläche und als Vermittlungsschicht ein Controller. Dies ist dem Entwurfsmuster Model-View-Controller nachempfunden. Grund hierfür ist die Austauschbarkeit der Oberfläche, die sich sowohl im Laufe der Zeit ändern kann, als auch von möglichen Kunden anders gefordert werden kann.

Ein weiterer Punkt ist das sammeln von Log-Daten<sup>30</sup>, welche zu jedem Zeitpunkt, in dem das Programm läuft, auf dem PC gespeichert werden müssen. Dies dient dem Nachvollziehen von Fehlern im Programm, aber auch in den Befehlen, die das Programm empfängt.

Neben dem Programm-Code muss auch zu jeder Funktion ein Java-Doc-Eintrag verfasst werden. Dieser dient der besseren Lesbarkeit und führt zudem dazu, dass beim Erweitern oder Debuggen des Programms eine Dokumentation zu den genutzten Schnittstellen vorhanden ist.

## **Maximale Anzahl an gleichzeitig vorhandenen Schlitten**

Die Anzahl der gleichzeitig bearbeitbaren Schlitten ist von Auftraggeber nicht begrenzt worden. Es müssen theoretisch unendlich viele Schlitten – und demnach auch Paletten – verarbeitet werden können.

## **Maximale Anzahl an gleichzeitigen Netzwerknachrichten**

Die Netzwerkschnittstelle an sich empfängt die Nachrichten seriell und erzeugt sofort jeweils einen Thread, der die Nachricht bearbeitet. Werden also mehrere Nachrichten in einem kurzen Zeitraum empfangen, so werden genauso viele Threads erstellt. Somit ist die Anzahl der maximal bearbeitbaren Nachrichten und die daraus folgenden Befehle nicht begrenzt. Das Betriebssystem kümmert sich um die Rechenzeit der einzelnen Threads. Es können, genau wie bei den Schlitten, theoretisch unendlich viele Befehle bearbeitet werden.

## **Anzahl Transportsysteme**

Die Anzahl der zusammenhängenden Systeme ist vom Auftraggeber auf genau eines begrenzt worden. Es ist nicht das Ziel der FDZ, mehrere unabhängige Transportsysteme zu simulieren, sonder nur jeweils eines zu jedem Zeitpunkt.

## **Portierbarkeit**

Das Programm ist für einen Standard-PC ausgelegt, auf dem Windows 7 installiert ist. Grundvoraussetzung ist auch, wie im Kapitel Plattform erläutert, dass Java in der Version 1.8 installiert ist. Es ist nicht vorgesehen, die Software portierbar auf andere Betriebssysteme oder Rechnerarchitekturen zu entwickeln, allerdings folgert diese Eigenschaft aus der Voraussetzung, dass die Simulation in Java programmiert wird. Somit ist die Portierbarkeit im Rahmen von Java-fähigen Systemen nur durch die Eigenschaften von Java begrenzt.

---

30 Siehe Kapitel Log

# Architekturgrundlagen

In diesem Kapitel werden die grundlegenden Architekturentscheidungen vorgestellt. Ziel ist es nicht, eine fertige Architektur zu entwickeln, sondern eine grobe Stütze für den weiteren Verlauf des Projektes zu bilden. Ein großer Teil der Entscheidungen beruht auch auf den wünschen der Auftraggeber<sup>31</sup>.

## Model-View-Controller

Nach Absprache mit dem Auftraggebern ist es das große Ziel dieses Projektes, für die Simulation ein großes Maß an Wiederverwendbarkeit und Modularität zu gewährleisten. Ein gewählter Schritt, um die Wiederverwendbarkeit der Grundarchitektur zu garantieren ist die Entwicklung des System durch das Entwurfsmuster Model-View-Controller. Dies ist nicht nur guter Stil in der Programmierung, es erlaubt dem Team auch, die gesamte Logik des Systems im Voraus zu implementieren und zu testen. Nach der Fertigstellung der Grundlage kann dann mit dem JavaFX Scene Builder<sup>32</sup> eine Grafische Oberfläche nach den Wünschen der Auftraggeber (dazu in späteren Kapiteln mehr) entworfen werden, die dann nur noch durch einem Controller mit der Logik verknüpft werden muss. Die Vorteile liegen auf der Hand, die Logik kann jederzeit abgekoppelt und an eine neue Oberfläche angebunden werden. Der Programmiercode wird somit auch besser wiederverwendbar und lesbarer.

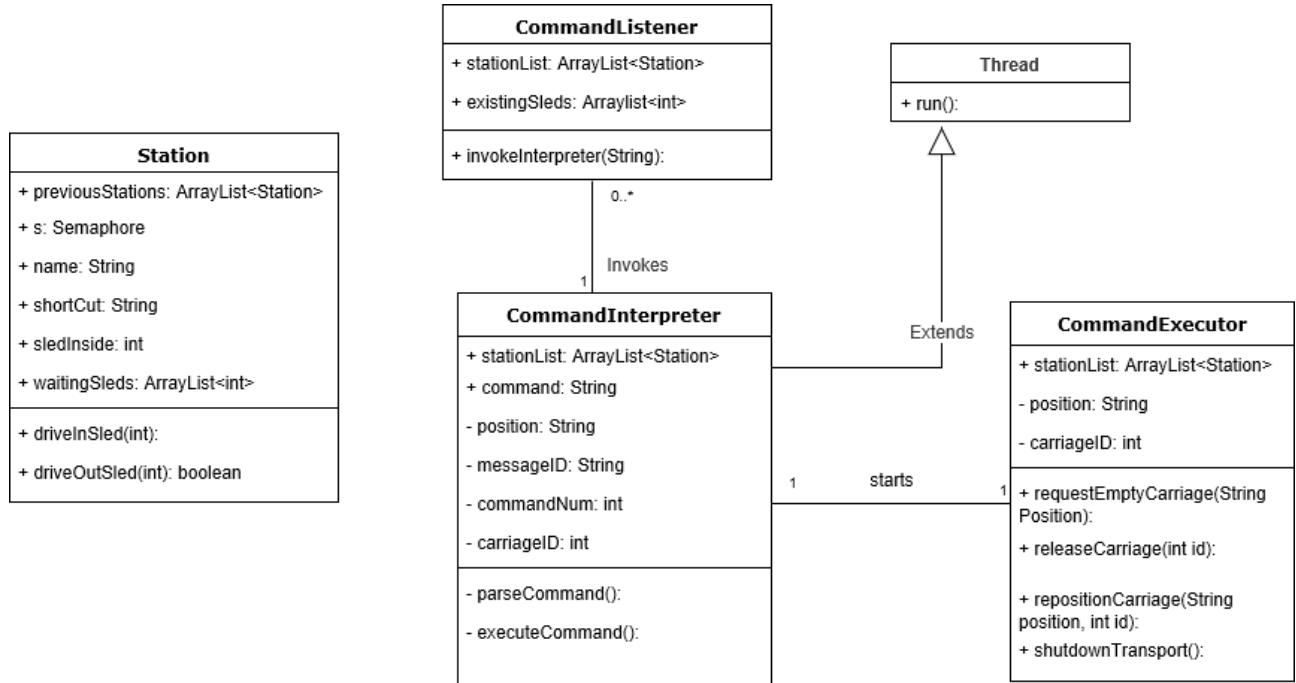
---

31 Anmerkung: Dieses Projekt wird agil durch die Vorgehensweise Scrum durchgeführt. Zu beachten ist deshalb, dass das Team sich im Vorhinein nur das Grobkonzept der Architektur herleiten kann. Dennoch sind früh im Projekt einige Leitfäden entworfen worden, nach denen sich die Entwicklern richten müssen.

32 JavaFX Scene Builder ist unter der Oracle BSD Lizenz veröffentlicht

# Klassendiagramm Model

Im folgenden wird schematisch dargestellt, in welche Schichten das System aufgeteilt wird:



*Abbildung 1: Klassendiagramm Model*

Zu sehen sind die Kernelemente der Klassenstruktur des Programms. Bekommt der CommandListener eine Nachricht, so startet er einen neuen CommandInterpreter. Dieser erbt von Thread und läuft deshalb ab diesem Zeitpunkt unabhängig weiter. Sobald der CommandInterpreter die Nachricht fertig geparsed hat, ruft er die Klasse CommandExecutor auf, welche, je nach benötigter Methode, das Kommando ausführt.

Die Klasse Station wird benötigt, um alle relevanten Informationen zur Station und den Verbindungen zu speichern. Eine Liste der Stationen wird an die Klassen übergeben, die sie für die Ausführung der Kommandos brauchen.

## Sequenzdiagramm Empfang einer Nachricht

Das folgende Sequenzdiagramm erläutert den Ablauf des Programms, wenn es eine Nachricht von der übergeordneten Komponente der FDZ empfangen hat:

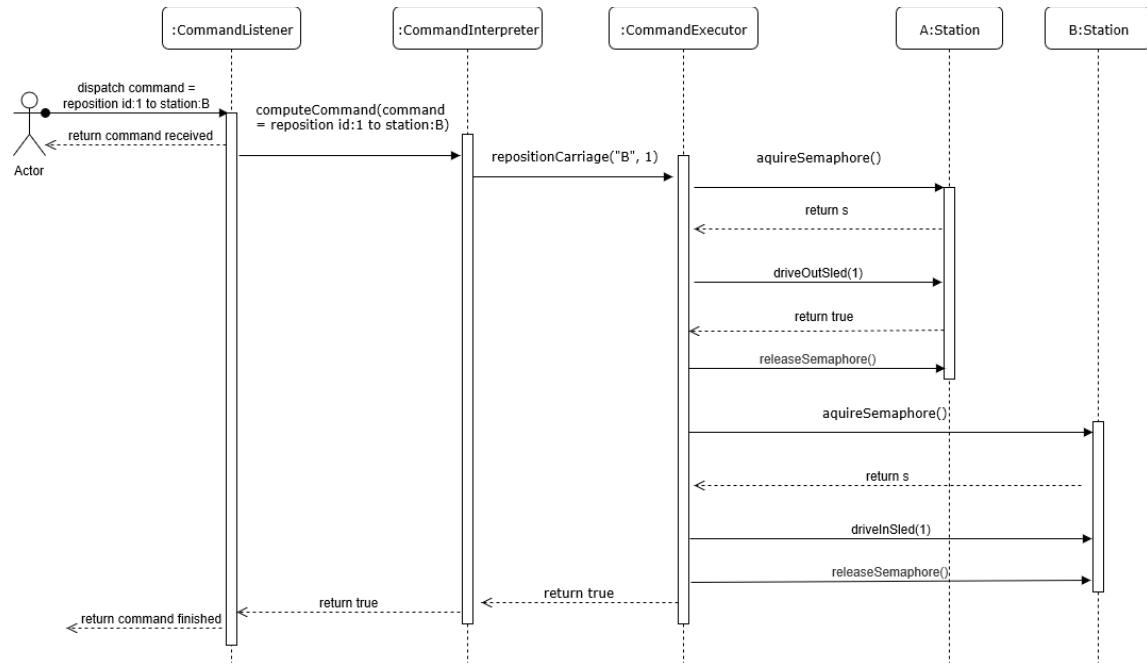


Abbildung 2: Sequenzdiagramm Abarbeitung

## Sequenzdiagramm Netzwerkschnittstelle

Im nächsten Sequenzdiagramm erkennt man, welche Kommunikation vom Auftraggeber durch die zu nutzenden Protokolle gefordert wurden:

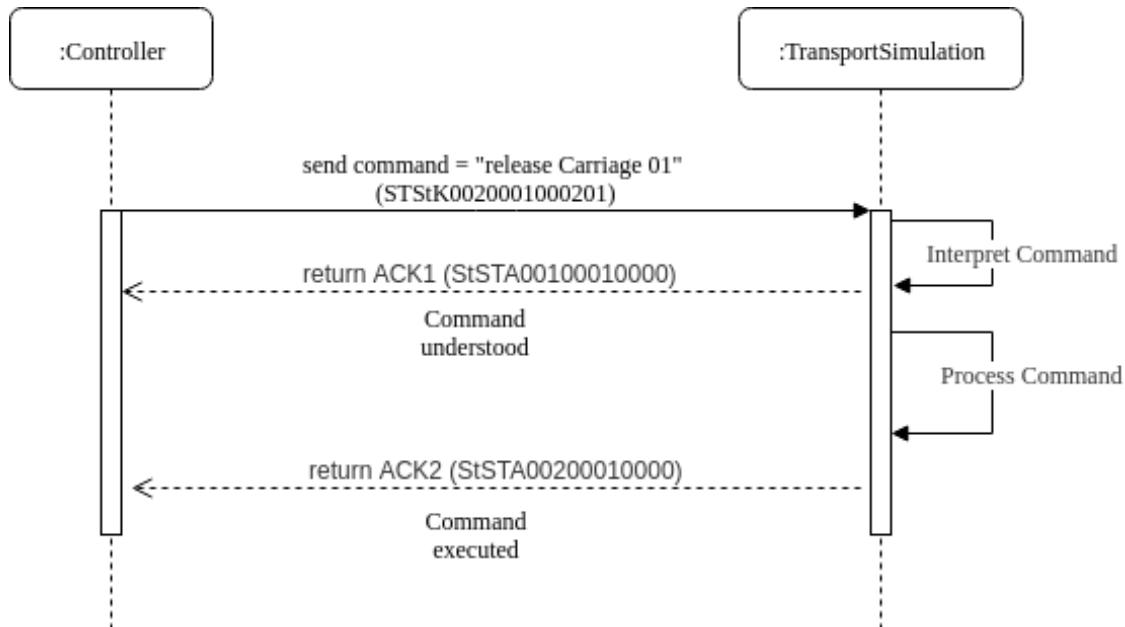


Abbildung 3: Sequenzdiagramm Netzwerkkommunikation

# Layout

In diesem Kapitel werden die Screendesigns vorgestellt, die nach Absprache mit dem Kunden das finale Design für die Entwicklung des Programms darstellen.

## Gesamtüberblick

Das folgende Screendesign zeigt die gesamte grafische Oberfläche, hier wird kein Element bedient oder angeklickt:

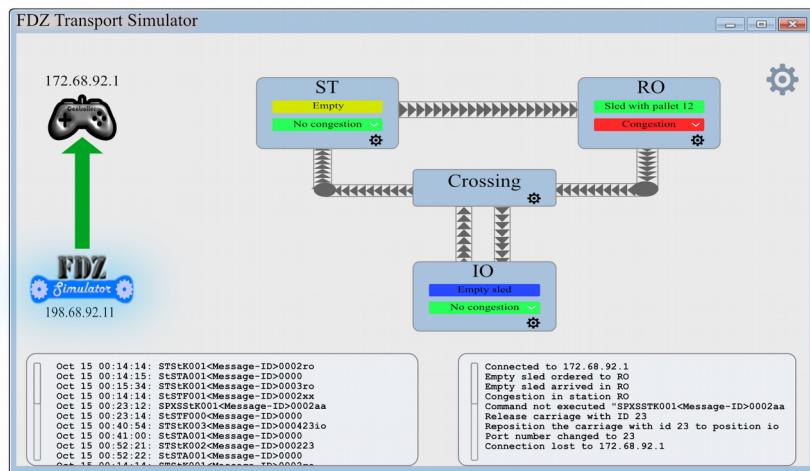


Abbildung 4: Screendesign Gesamt

Wie in den Anforderungen beschrieben werden folgende Punkte angezeigt:

- Verbindungsstatus

Links im Bild, das untere Logo beschreibt die Simulationssoftware, das obere die übergeordnete Instanz, den Controller.

- Stationen

In den Grau-Blauen Vierecken zu sehen ist die Abkürzung der Station, darunter der Schlitten/ die Palette, die darin ist und der Status der Station mit einem Drop-Down-Menü, das eine Liste aller Schlitten/Paletten enthält.

- Kreuzungen

Sind im Bild mit Crossing gekennzeichnet, tragen zur besseren Übersicht bei.

- Log

Links unten im Bild ist ein Textfeld mit den Logmeldungen, die das System parallel auch in einer externen Datei speichert.

- Status

Rechts unten im Bild ist das Status-Textfeld, darin enthalten die menschenlesbaren Meldungen über die Aktion, die die Simulation aktuell durchführt.

# Optionen

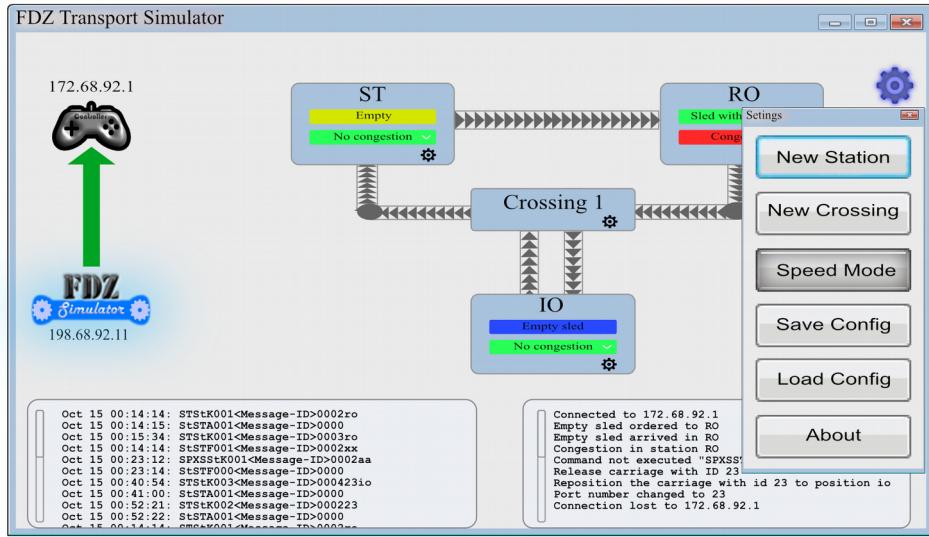


Abbildung 5: Screendesign Optionen

Rechts oben im Bild ist ein Zahnrad zu sehen. Es symbolisiert das Optionen-Menü, welches geöffnet so wie im oberen Bild aussieht.

## Konfiguration der Stationen

Das folgende Bild zeigt die Möglichkeit, Stationen zu konfigurieren:

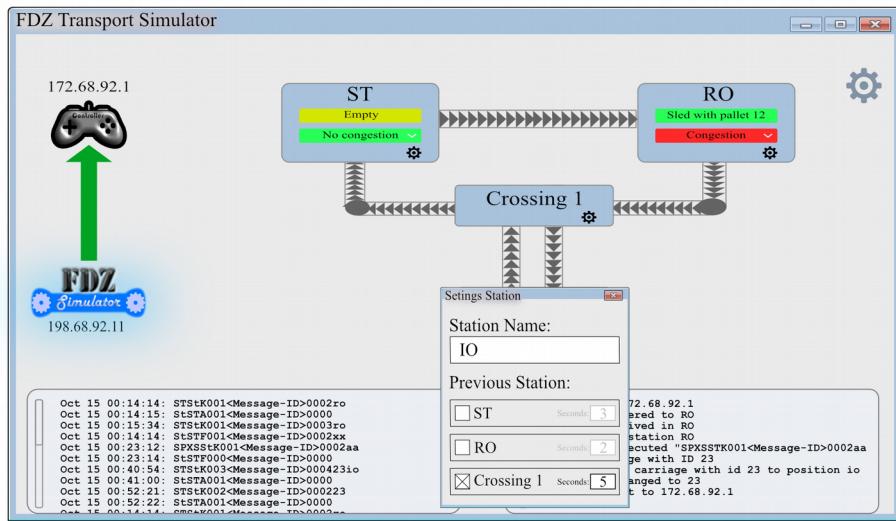


Abbildung 6: Screendesign Stationen konfigurieren

Das Zahnrad in den rechten unteren Ecken der Stationen und Kreuzungen öffnet ein Fenster, welches die Konfiguration des ausgewählten Objektes ermöglicht. Zu erkennen sind eine Liste der Stationen, von denen die ausgewählte erreichbar ist, und ein Textfeld, dass die benötigte Zeit für die Strecke von dieser Station entgegen nimmt.

# Netzwerkkonfiguration

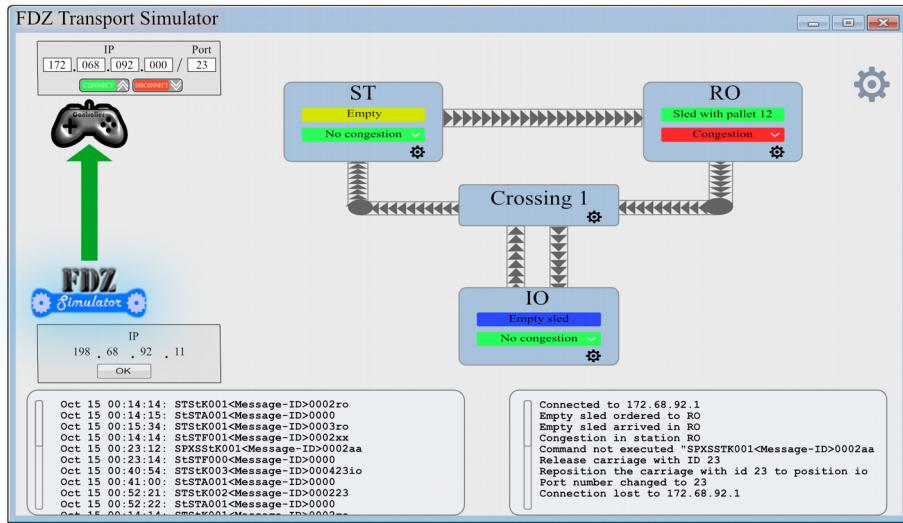


Abbildung 7: Screendesign Netzwerkkonfiguration

Das obere Bild zeigt die Möglichkeit, per Mausklick auf die IP-Adressen der zwei Elemente links im Bild die Adressen und Port-Einstellungen zu konfigurieren. Bei der Konfiguration kann dann auch die Verbindung hergestellt oder unterbrochen werden.

## Station-Attribute

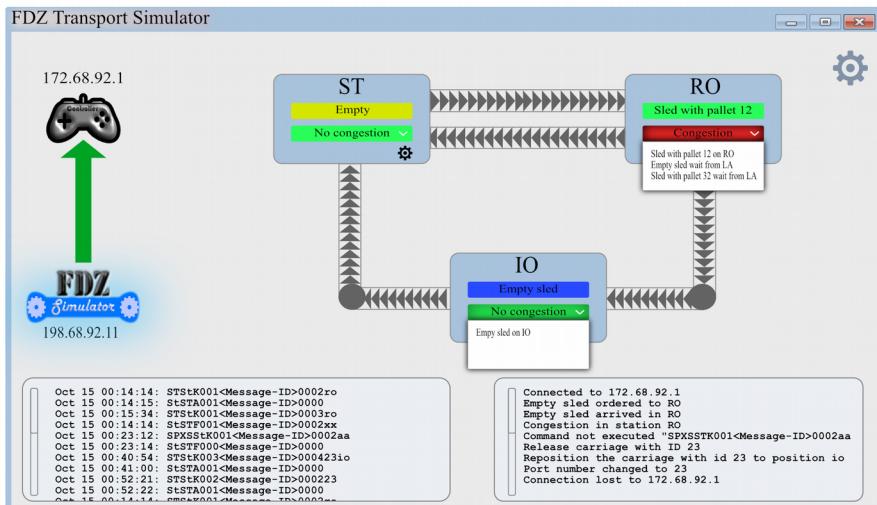


Abbildung 8: Screendesign Attribute

Klickt man auf eines der Drop-Down-Symbole in den Attributen der Stationen, so bekommt man eine Liste angezeigt, die alle Schlitten oder Paletten enthält, die in der Station sind oder an ihr im Stau stehen.